Struktury i działania na plikach w C# - skrypt:

Zajęcia laboratoryjne przeprowadzane są na komputerach z systemem operacyjnym Windows 7 z wykorzystaniem oprogramowania Visual Studio 2010 w wersji Ultimate. Poniżej omówiono kilka często wykorzystywanych skrótów:
VS – Visual Studio
LPM – Lewy przycisk myszy
PPM – Prawy przycisk myszy

Zagadnienie 1: Struktury
Struktury to lżejsza, bardziej wydajna wersja klas. Dzięki nim można zrealizować właściwie to samo, z drobnymi wyjątkami i mniejszą swobodą. Podstawową różnicą pomiędzy nimi jest sposób ich implementacji. W przypadku klas nowe ich instancje odkładane są na tzw. stertę, natomiast w przypadku struktur ich instancje odkładane są na stos. Dalej, działając na strukturach działamy bezpośrednio na nich, a nie jak w przypadku klas na referencjach do nich. Powoduje to, że przekazując je jako parametry wywołania funkcji przekazywane są przez wartość a nie przez referencję.
Powodem wydzielania struktur od klas są aplikacje gdzie liczy się wydajność, a używa się ogromnych ilości prostych obiektów. Wówczas zdecydowanie lepiej jest zamodelować obiekty w prostszy ale bardziej efektywny sposób czyli poprzez implementację ich jako struktur.

Przykład:

```
struct Car
{
        private string color;

        public Car(string color)
        {
                this.color = color;
        }

        public string Describe()
        {
                return "This car is " + Color;
        }

        public string Color
        {
                get { return color; }
                set { color = value; }
        }
}
```

Struktury, a klasy:
- słowo kluczowe struct a nie class
- pola nie mogą być inicjalizowane
- w konstruktorze wszystkie pola muszą zostać zainicjalizowane

- struktura może używać domyślnego konstruktora ale jak już jest zdefiniowany przez programistę to wszystkie pola muszą być inicjalizowane
- nowy konstruktor nie może być bezparametrowy
- struktury nie mogą dziedziczyć i nie mogą być dziedziczone ani przez inne struktury ani przez klasy
- struktury mogą implementować interfejsy

Zagadnienie 2: Czytanie i zapisywanie do plików tekstowych

W tym rozdziale zostaną omówione zagadnienia związane z zapisywaniem oraz czytaniem prostych plików z wykorzystaniem języka C#. W tym celu wykorzystywana jest predefiniowana klasa File należąca do przestrzeni nazw System.IO, która implementuje niemalże wszystkie proste operacje na plikach.

Na początku stworzony zostanie prosty edytor tekstowy, który pozwala na odczytanie podanego pliku tekstowego, a następnie zapisanie do niego nowej zawartości (jednej linii).

```
if (File.Exists("test.txt"))
{
        string content = File.ReadAllText("test.txt");
        Console.WriteLine("Current content of file:");
        Console.WriteLine(content);
}
Console.WriteLine("Please enter new content for the file:");
string newContent = Console.ReadLine();
File.WriteAllText("test.txt", newContent);
```

Jak widać klasa File została wykorzystana w trzech miejscach:

- sprawdzenie czy podany plik istnieje
- wykorzystanie metody klasy File ReadAllText do odczytania danych z pliku
- wywołanie metody WriteAllText w celu zapisania nowej zawartości pliku

Istotnym jest, że w podanym przykładzie nie podawana jest pełna ścieżka do pliku. Podana jest jedynie nazwa pliku co określa, że plik na którym wykonujemy operacje znajduje się w katalogu w którym znajduje się plik wykonywalny naszego programu. Przedstawiony powyżej program, wykonuje swoje zadanie, natomiast nadpisuje za każdym razem zawartość pliku tekstowego. Aby dopisywać nową zawartość do istniejącego pliku ostatnią linijkę, wyżej przedstawionego kodu należy podmienić na:

```
File.AppendAllText("test.txt", newContent);
```

Po uruchomieniu kodu z powyższą zmianą, podawany za każdym razem ciąg znaków będzie dopisywany do poprzednio wprowadzonej zawartości. Kolejnym krokiem jest umożliwienie wprowadzania kilku linii tekstu za jednym razem do edytowanego pliku. W tym celu należy podmienić odpowiedni linie kodu na następujace:

```
...
Console.WriteLine("Please enter new content for the file - type exit and press enter to finish editing:");
string newContent = Console.ReadLine();
```

```
{
        File.AppendAllText("test.txt", newContent + Environment.NewLine);
        newContent = Console.ReadLine();
}
```

Pozwoli to na edytowanie pliku tekstowego, do czasu aż użytkownik nie poda ciągu znaków „exit”, który oznacza zakończenie edycji pliku. Na uwagę zasługuje dodawanie do zawartości tekstowej podawanej przez użytkownika wartości Environment.NewLine, która pozwala na dodanie znaku końca linii odpowiedniego środowiska w którym działa aplikacja (ciąg znaków zawierający "\r\n" dla systemów spoza rodziny Unix, lub ciąg znaków "\n" dla systemów typu Unix).
Przedstawiony w w powyższym przykładzie sposób edycji plików tekstowych jest bardzo prosty, natomiast powoduje otwieranie i zamykanie pliku za kazdym razem kiedy wykonujemy na nim dowolne działanie. W praktyce aplikacja taka jest poprawna, natomiast działa w sposób nieefektywny – tracone jest czas na otwieranie oraz zamykanie pliku za kazdym razem, kiedy ma nastąpic odczyt lub zapis do pliku. Poniżej przedstawiono kod pozwalający zapobiec niepotrzebnemu marnowaniu zasobów oraz czasu procesora przez aplikację edytora tekstu.

```
Console.WriteLine("Please enter new content for the file - type exit and press enter to finish editing:");
using(StreamWriter sw = new StreamWriter("test.txt"))
{
   string newContent = Console.ReadLine();
   while(newContent != "exit")
   {
      sw.Write(newContent + Environment.NewLine);
      newContent = Console.ReadLine();
   }
}
```

W powyższym kodzie zwróć uwagę na słowo kluczowe using.

Zagadnienie 2: Zarządzanie systemem plików
Poza prostymi mechanizmami zapisywania oraz odczytywania plików tekstowych (omówione wcześniej) czy binarnych, z poziomu klas .NET możliwe jest zarządzanie systemem plików z zakresie: tworzenia, usuwanie, zmiany nazwy czy kopiowania plików.

Usuwanie pliku:

```
if(File.Exists("test.txt"))
{
   File.Delete("test.txt");
   if(File.Exists("test.txt") == false)
      Console.WriteLine("File deleted...");
}
else
   Console.WriteLine("File test.txt does not yet exist!");
Console.ReadKey();
```

Usuwanie katalogu (pustego):

```
if(Directory.Exists("testdir"))
{
   Directory.Delete("testdir");
   if(Directory.Exists("testdir") == false)
      Console.WriteLine("Directory deleted...");
}
else
   Console.WriteLine("Directory testdir does not yet exist!");
Console.ReadKey();
```

Kasowanie katalogu rekursywnie:

```
Directory.Delete("testdir", true);
```

Zmiana nazwy pliku:

```
if(File.Exists("test.txt"))
{
   Console.WriteLine("Please enter a new name for this file:");
   string newFilename = Console.ReadLine();
   if(newFilename != String.Empty)
   {
      File.Move("test.txt", newFilename);
      if(File.Exists(newFilename))
      {
         Console.WriteLine("The file was renamed to " + newFilename);
         Console.ReadKey();
      }
   }
}
```

Zmiana nazwy katalogu:

```
if(Directory.Exists("testdir"))
{
   Console.WriteLine("Please enter a new name for this directory:");
   string newDirName = Console.ReadLine();
   if(newDirName != String.Empty)
   {
      Directory.Move("testdir", newDirName);
      if(Directory.Exists(newDirName))
      {
         Console.WriteLine("The directory was renamed to " + newDirName);
         Console.ReadKey();
      }
   }
}
```

Tworzenie nowego katalogu:

```
Console.WriteLine("Please enter a name for the new directory:");
string newDirName = Console.ReadLine();
if(newDirName != String.Empty)
```

```
{
   Directory.CreateDirectory(newDirName);
   if(Directory.Exists(newDirName))
   {
      Console.WriteLine("The directory was created!");
      Console.ReadKey();
   }
}
```

Zagadnienie 3: Pobieranie informacji o plikach i katalogach

Wykorzystywane wcześniej klasy File oraz Directory, służą do manipulacji plikami oraz katalogami. Czasami jednak, może być potrzebne działania na atrybutach plików oraz katalogów takich jak prawa dostępu czy zawartość w przypadku katalogów. Do tego celu służą do tego klasy: FileInfo oraz DirectoryInfo.

Pobieranie informacji na temat pliku:

```
FileInfo fi = new FileInfo(@"C:\itsoa\tomcat.rar");
if(fi != null)
        Console.WriteLine(String.Format("Information about file: {0}, {1} bytes, last
modified on {2} - Full path: {3}", fi.Name, fi.Length, fi.LastWriteTime, fi.FullName));
Console.ReadKey();
```

Pobieranie informacji na temat katalogu:

```
DirectoryInfo di = new DirectoryInfo(@"C:\itsoa\tomcat\");
DirectoryInfo[] subDirs = di.GetDirectories();
if (subDirs.Length > 0)
{
        Console.WriteLine("Directories:");
        foreach (DirectoryInfo subDir in subDirs)
        {
                Console.WriteLine("   " + subDir.Name);
        }
}
if (di != null)
{
        FileInfo[] subFiles = di.GetFiles();
        if (subFiles.Length > 0)
        {
                Console.WriteLine("Files:");
                foreach (FileInfo subFile in subFiles)
                {
                        Console.WriteLine("   " + subFile.Name + " (" + subFile.Length + "
bytes)");
                }
        }
Console.ReadKey();
}
```

Permutacje powyższego przykładu:

Pobierz pliki z rozszerzeniem .exe.
```
FileInfo[] subFiles = di.GetFiles("*.exe");
```

Pobieranie katalogów zawierających ciąg znaków test w dowolnym miejscu w nazwie.
```
DirectoryInfo[] subDirs = di.GetDirectories("*test*");
```

Poszukiwanie plików z rozszerzeniem exe rekursywnie w katalogu.
```
FileInfo[] subFiles = di.GetFiles("*.exe", SearchOption.AllDirectories);
```

Poszukiwanie plików z rozszerzeniem exe w aktualnym katalogu.
```
FileInfo[] subFiles = di.GetFiles("*.exe", SearchOption.TopDirectoryOnly);
```

Zagadnienie 4: Wykorzystanie protokołu TCP/IP

Inner Process Comunication to sposób na łączenie i komunikację dwóch lub więcej maszyn w celu wymiany danych pomiędzy nimi. Zazwyczaj do tego celu wykorzystuje się protokół TCP/IP. Poniżej przedstawiony jest kod pozwalający na sprawdzenie jaki numer IP posiada komputer.

```
string name = (args.Length < 1) ? Dns.GetHostName() : args[0];
try
{
        IPAddress[] addrs = Dns.Resolve(name).AddressList;
        foreach (IPAddress addr in addrs)
                Console.WriteLine("{0}/{1}", name, addr);
}
catch (Exception e)
{
        Console.WriteLine(e.Message);
}
Console.ReadLine();
```

Tworzenie aplikacji klienta:

```
try
{
        TcpClient tcpclnt = new TcpClient();
        Console.WriteLine("Connecting.....");

        tcpclnt.Connect("127.0.0.1", 8001);

        Console.WriteLine("Connected");
        Console.Write("Enter the string to be transmitted : ");

        String str = Console.ReadLine();
        Stream stm = tcpclnt.GetStream();

        ASCIIEncoding asen = new ASCIIEncoding();
        byte[] ba = asen.GetBytes(str);
        Console.WriteLine("Transmitting.....");

        stm.Write(ba, 0, ba.Length);
```

```
        byte[] bb = new byte[100];
        int k = stm.Read(bb, 0, 100);

        for (int i = 0; i < k; i++)
                Console.Write(Convert.ToChar(bb[i]));

        tcpclnt.Close();
}
catch (Exception e)
{
        Console.WriteLine("Error..... " + e.StackTrace);
}
Console.ReadLine();
```

Tworzenie aplikacji serwera:

```
try
{
        IPAddress ipAd = IPAddress.Parse("127.0.0.1");

        TcpListener myList = new TcpListener(ipAd, 8001);

        myList.Start();

        Console.WriteLine("The server is running at port 8001...");
        Console.WriteLine("The local End point is  :" +
                          myList.LocalEndpoint);
        Console.WriteLine("Waiting for a connection.....");

        Socket s = myList.AcceptSocket();
        Console.WriteLine("Connection accepted from " + s.RemoteEndPoint);

        byte[] b = new byte[100];
        int k = s.Receive(b);
        Console.WriteLine("Recieved...");
        for (int i = 0; i < k; i++)
                Console.Write(Convert.ToChar(b[i]));

        ASCIIEncoding asen = new ASCIIEncoding();
        s.Send(asen.GetBytes("The string was recieved by the server."));
        Console.WriteLine("\nSent Acknowledgement");

        s.Close();
        myList.Stop();

}
catch (Exception e)
{
        Console.WriteLine("Error..... " + e.StackTrace);
}
Console.ReadLine();
```

Linki:

[1] MSDN: http://msdn.microsoft.com/
[2] C# Practical Learning: http://www.functionx.com/csharp/
[3] C# for beginners: http://www.csharphelp.com/2006/12/c-tutorial-for-beginners/
[4] C# tutorial: http://csharpcomputing.com/Tutorials/
[4] C# tutorials: http://csharp.net-tutorials.com/